МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедры МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студентка гр. 8304 _____ Мельникова О.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы

Ознакомиться с основными понятиями и структурой иерархических списков, получить навыки программирования с использованием базовых функций рекурсивной обработки списков на языке программирования C++.

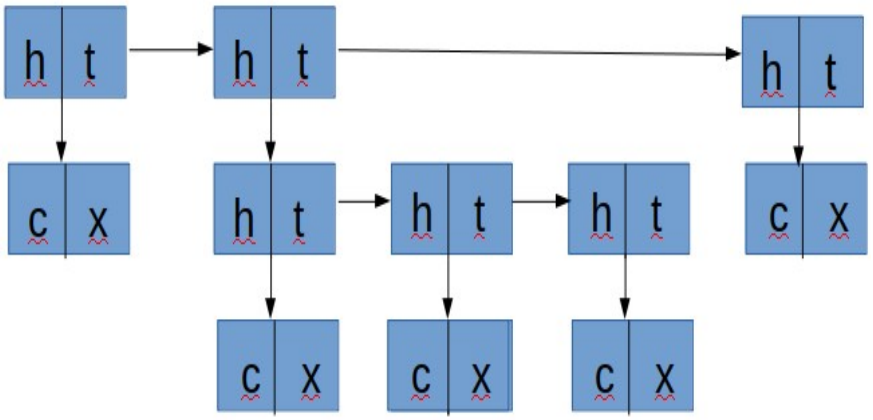
Задание

Вариант 6

Проверить иерархический список на наличие в нем заданного элемента (атома) x.

Представление иерархического списка

Иерархический список был представлен в памяти следующим образом:



Пример для иерархического списка (a(bcd)e)

Здесь h - *s_expr *hd*, t - *s_expr *tl*, c - *bool tag* = 1, x – *base value*.

```
typedef char base;
struct s_expr;
struct two_ptr{
    s_expr *hd;
    s_expr *tl;
};
struct s_expr{
    bool tag;
    union{
        base value;
        two_ptr my_pair;
    }node;
};
typedef s_expr *lisp;
```

Считывание иерархического списка.

Было реализовано считывание из файла и считывание из консоли. Считывание из файла происходит тогда, когда есть три правильных (это проверяется с помощью регулярных выражений) агрумента командной строки, содержащих названия txt файлов с исходным списком, символом для поиска и файл для записи результата. Сокращенная запись списка заносится в `string str`, после чего она обрабатывается функциями `read_lisp`, `read_s_expr`, `read_seq`, и создается иерархический список `lisp mainLisp`.

Функция `read_lisp`

Функция принимает иерархический список и строку (с записью этого списка) по ссылкам. Процедура позволяет пропустить пробелы, которые могут стоять в начале. С помощью метода `.substr(1)` строка теряет первый символ и становится на 1 знак короче. После удаления первых пробелов (если они были) вызывается функция `read_s_expr`, куда передается новая строка, список и первый символ строки.

Функция `read_s_expr`

Процедура создает атом, если не было ошибок при вводе, и текущий символ это не скобка (происходит вывзов функции `make_atom`).

Если же символ – это окрывающаяся скобка, то вызывается следующая функция `read_seq` принимающая список и строку.

Функция `read_seq`

Функция вызывается при начале подсписка, кроме проверок ошибок она вызывает функцию `read_s_expr`, которая в случае наличия подсписков заполнит и их, в нее передается указатель `p1` на подсписок. Вызов функции `read_seq` позволяет уакзатель на

следующий элемент в иерархическом списке p2. Функция cons с использованием p1 и p2 заполяет в структуре элемента списка типа s_expr пару two_ptr my_pair; указателей на подсписок и следующий элемент вышестоящего списка.

Проверка на наличие атома в списке.

Проверку осуществляет функция in_lisp которая принимает символ, а также иерархический список, возвращает функция значение типа bool:

1 - атом найден, 0 – атом не был найден.

Если список не пуст происходит разделение: если элемент – атом, то проверяется значение value, при совпадении возвращается true; если элемент – подсписок, то сначала вызывается in_lisp(x, head(l)) , то есть максимально углубляемся, а затем вызывается in_lisp(x, tail(l)), то есть постепенно поднимаемся вверх проходя все элементы на уровнях. Таким образом проверяется весь иерархический список, в случае обнаружения нужного атома возвращается true, если в итоге атом не был найден – происходит возврат false.

Базовые функции обработки иерархических списков

Оъявление функции	Назначение
lisp head(const lisp my_list);	Возвращает указатель на подсписок
lisp tail(const lisp my_list);	Возвращает указатель на следующий элемент
bool isAtom(const lisp my_list);	Возвращает 1 или 0 в зависимости элемент атом или подсписок

`bool isNull(const lisp my_list);`	Проверяет не пуст ли список
`lisp cons(const lisp h, const lisp t);`	В структуре элемента списка типа `s_expr` заполняет пару `my_pair` указателей на подсписок и следующий элемент вышестоящего списка, а также устанавливает в `tag` значение false
`lisp make_atom(const base x);`	В структуре элемента списка типа `s_expr` устанавливает в `tag` значение true, заполняет значение `value`.
`void destroy(lisp my_list);`	Удаляет весь иерархический список из памяти

Тестирование

<table>
<tr><td>Содержимое файла lisp.txt:
(abdg(tr(tg)mb)qw)</td><td rowspan="2">Содержимое файла Test1.txt:
b был найден в иерархическом списке</td></tr>
<tr><td>Содержимое файла symb.txt:
b</td></tr>
<tr><td>Содержимое файла lisp.txt:
(gksvkdbkj(jfhv)k)</td><td rowspan="2">Содержимое файла Test2.txt:
q не был найден в иерархическом списке</td></tr>
<tr><td>Содержимое файла symb.txt:
q</td></tr>
<tr><td>Содержимое файла lisp.txt:
((((((kgh)</td><td rowspan="2">Вывод в консоль:
Неправильный ввод списка</td></tr>
<tr><td>Содержимое файла symb.txt:
z</td></tr>
<tr><td>Содержимое файла lisp.txt:
(abdg(trmb)qw))))</td><td rowspan="2">Вывод в консоль:
Неправильный ввод списка</td></tr>
<tr><td>Содержимое файла symb.txt:
m</td></tr>
<tr><td>Содержимое файла lisp.txt:
)jhdk(jd)</td><td rowspan="2">Вывод в консоль:
Неправильный ввод списка</td></tr>
<tr><td>Содержимое файла symb.txt:
b</td></tr>
</table>

Вывод

В данной работе было создана программа, которая с использованием базовых функций рекурсивной обработки списков проверяет на наличие заданного элемента (атома) в данном иерархическом списке.

Исходный код программы

```
#include <iostream>
#include <fstream>
#include <string>
#include <regex.h>
#include <string.h>

typedef char base;
struct s_expr;
struct two_ptr{
    s_expr *hd;
    s_expr *tl;
};
struct s_expr{
    bool tag;
    union{
        base value;
        two_ptr my_pair;
    }node;
};
typedef s_expr *lisp;
```

```
using namespace std;



void read_lisp( lisp& my_list, std::string
&str);
void read_s_expr(base symb, lisp& my_list,
std::string &str);
void read_seq(lisp& my_list, std::string
&str);

bool in_lisp(base x, const lisp l);

lisp head(const lisp my_list);
lisp tail(const lisp my_list);
bool isAtom(const lisp my_list);
bool isNull(const lisp my_list);
lisp cons(const lisp h, const lisp t);
lisp make_atom(const base x);
void destroy(lisp my_list);

//ifstream file;

int main(int argc, char *argv[])
{
    lisp mainLisp;
    char symb = ' ';
    string str;
```

```
    int flag;
    if((argv[1]!=NULL)&&(argv[2]!
=NULL)&&(argv[3]!=NULL)){
        regex_t regex;
        int reti1;
        int reti2;
        int reti3;
        reti1 = regcomp(&regex, ".\\.txt", 0);
        reti1 = regexec(&regex, argv[1], 0,
NULL, 0);
        reti2 = regcomp(&regex, ".\\.txt", 0);
        reti2 = regexec(&regex, argv[2], 0,
NULL, 0);
        reti3 = regcomp(&regex, ".\\.txt", 0);
        reti3 = regexec(&regex, argv[3], 0,
NULL, 0);
        if ((!reti1)&&(!reti2)&&(!reti3)){
            ifstream fin(argv[1], ios::in);
            std::getline(fin, str);
            read_lisp(mainLisp, str);
            str=str.substr(1);
            if(!str.empty()){
                cerr<<"Неправильный ввод
списка"<<endl;
                exit(1);
            }
            ifstream file(argv[2], ios::in);
            file>>symb;
            if (symb==' '){
```

```
                cerr<<"Неправильный ввод
символа"<<endl;
                destroy(mainLisp);
                exit(1);
            }
        ofstream fout(argv[3], ios::out);
        fout<< symb;
        if(in_lisp(symb, mainLisp)){
            fout<<" был найден в
иерархическом списке.";
            }else{
                fout<<" не был найден в
иерархическом списке.";
            }
            destroy(mainLisp);
            return 0;
        }else{
            cout<<"Некорректные названия
файлов (расширение txt)"<<endl;
            return 0;
        }
    }else{
        cout<<"\nВы не ввели или ввели
неправильно аргументы командной строки.\
nПервый аргумент командной строки – файл с
расширением txt, из которого считывается
список.\nВторой аргумент – файл с расширением
txt, в котором записан символ.\nТретий
аргумент – файл с расширением txt, куда будет
```

```
записан результат.\nДля продолжения введите 1,
иначе 0."<<endl;

        cin>>flag;
        if(flag==0) return 0;
        cout<<"Введите в консоль элементы
списка!"<<endl;
        cin>>str;
        read_lisp(mainLisp, str);
        str=str.substr(1);
        if(!str.empty()){
            cerr<<"Неправильный ввод
списка"<<endl;
            exit(1);
        }
        cout<<"Введите атом!"<<endl;
        cin>>symb;
    }
    if (mainLisp==NULL) exit(1);

    if(in_lisp(symb, mainLisp)){
        cout<<"Атом был найден в иерархическом
списке."<<endl;
    }else{
        cout<<"Атом не был найден в
иерархическом списке."<<endl;
    }
    destroy(mainLisp);
    return 0;
```

```
}

bool in_lisp(base x, const lisp l)
{
    if (l != NULL)
    {
        if (isAtom(l))
        {
            if (x == l->node.value) return
true;
        }
        else
        {
            if (in_lisp(x, head(l))) return
true;
            if (in_lisp(x, tail(l))) return
true;
        }
    }
    return false;
}
```

```
    x = str[0];
    }
    read_s_expr(x,my_list, str);
}

void read_s_expr(base symb, lisp &my_list,
std::string &str){
    if((symb == ')') || (symb == '\0')){
    cerr<<"Неправильный ввод списка"<<endl;
    destroy(my_list);
    exit(1);
    }
    else
        if(symb != '(') my_list =
make_atom(symb);
        else read_seq(my_list, str);
}

void read_seq(lisp& my_list, std::string &str)
{
    str = str.substr(1);
    base x = str[0];
    lisp p1,p2;
    if(x=='\0'){
    cerr<<"Неправильный ввод списка"<< endl;
    destroy(my_list);
    exit(1);
    }
```

```
    else{
        while(x==' '){
             str = str.substr(1);
             x = str[0];
         }
        if(x==')') my_list = NULL;
        else{
            read_s_expr(x,p1, str);
            read_seq(p2, str);
            my_list = cons(p1, p2);
        }
    }
}

lisp head(const lisp my_list){
    if(my_list != NULL)
        if(!isAtom(my_list)) return my_list-
>node.my_pair.hd;
        else{cerr<<"Error: Head(atom)\n";
exit(1);}
    else{cerr<<"Error:Head(nil)\n"; exit(1);}
}

lisp tail(const lisp my_list){
    if(my_list != NULL)
        if(!isAtom(my_list)) return my_list-
>node.my_pair.tl;
```

```
        else{cerr<<"Error: Tail(atom)\n";
exit(1);}

    else{cerr<<"Error: Tail(nil)\n"; exit(1);}
}

bool isAtom(const lisp my_list){
    if(my_list==NULL) return false;
    else return(my_list->tag);
}

bool isNull(const lisp my_list){
    return my_list==NULL;
}

lisp cons(const lisp h, const lisp t){
    lisp p;
    if(isAtom(t)){
        cerr<<"Error: cons(*, atom)\n";
exit(1);
    }else{ p = new s_expr;
        if(p==NULL){cerr<< "Memory Error\n";
exit(1);}
            else{
                p->tag = false;
                p->node.my_pair.hd = h;
                p->node.my_pair.tl = t;
                return p;
```

```
            }
    }
}

lisp make_atom(const base x){
    lisp my_list;
    my_list = new s_expr;
    my_list->tag = true;
    my_list->node.value = x;
    return my_list;
}

void destroy(lisp my_list){
    if( my_list != NULL){
        if(!isAtom(my_list)){
            destroy(head(my_list));
            destroy(tail(my_list));
        }
    delete my_list;
    };
}
```